Lab 1: Input/Output Interfacing and FSMs

Introduction

In this first lab, you will be using IAR Embedded Workbench and the TIVA LaunchPad to complete some simple tasks, such as blinking an LED.

By the end of this lab, you will:

- Be familiar with developing software using IAR Embedded Workbench
- Have experience programming the TM4C123 board using C code
- Understand the basics of general-purpose input/output (GPIO)
- Have experience using GPIO to control and read from external peripherals
- Have experience extracting important information from datasheets
- Have designed a finite state machine (FSM) controller in C

TM4C123GXL board

Figure 1 gives an overview of the layout of the TIVA TM4C123G LaunchPad. Both the user manual and the datasheet (found on Canvas) have much more information about the features and functionality of the board.

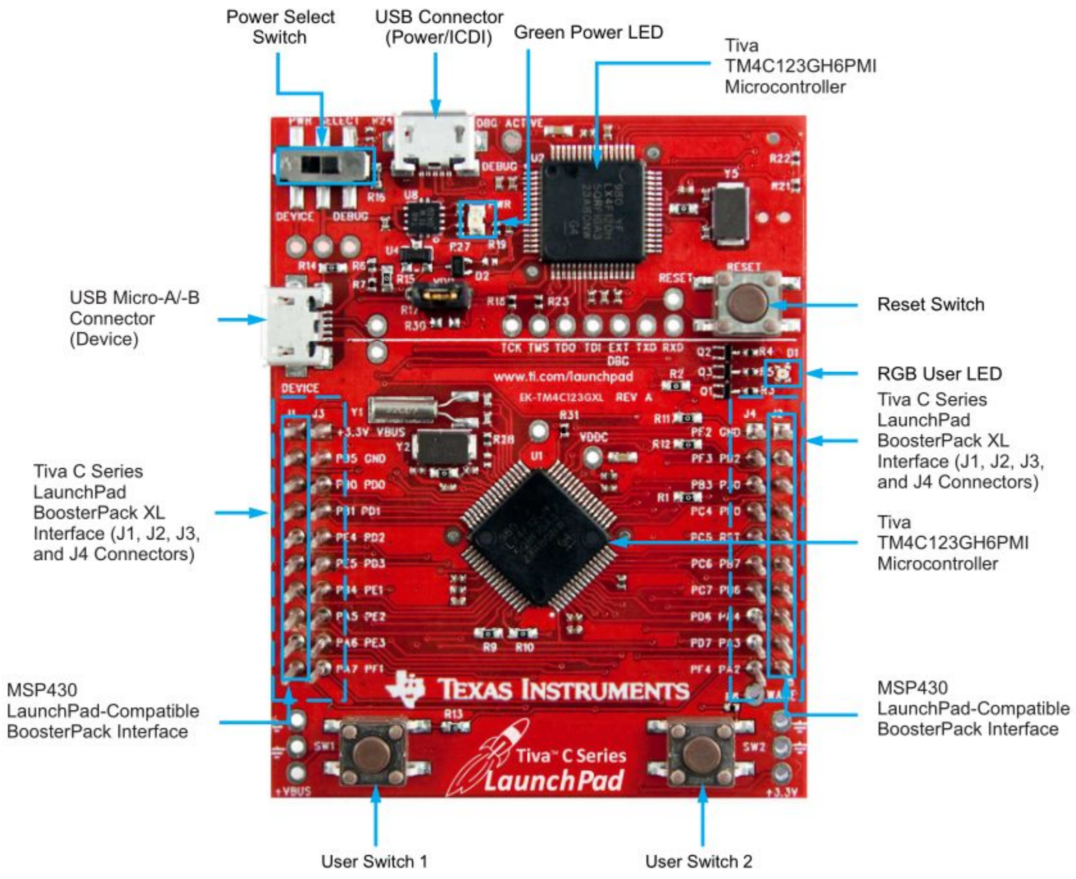

Power Select Switch
USB Connector (Power/ICDI)
Green Power LED
Tiva TM4C123GH6PMI Microcontroller
USB Micro-A/-B Connector (Device)
Reset Switch
RGB User LED
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva TM4C123GH6PMI Microcontroller
MSP430 LaunchPad-Compatible BoosterPack Interface
MSP430 LaunchPad-Compatible BoosterPack Interface
User Switch 1
User Switch 2

Figure 1. The Tiva LaunchPad

We'll start with lighting up the RGB User LED on the right. This consists of three LEDs – Red, Green, and Blue – connected to the processor through port F, as shown in the following figure:

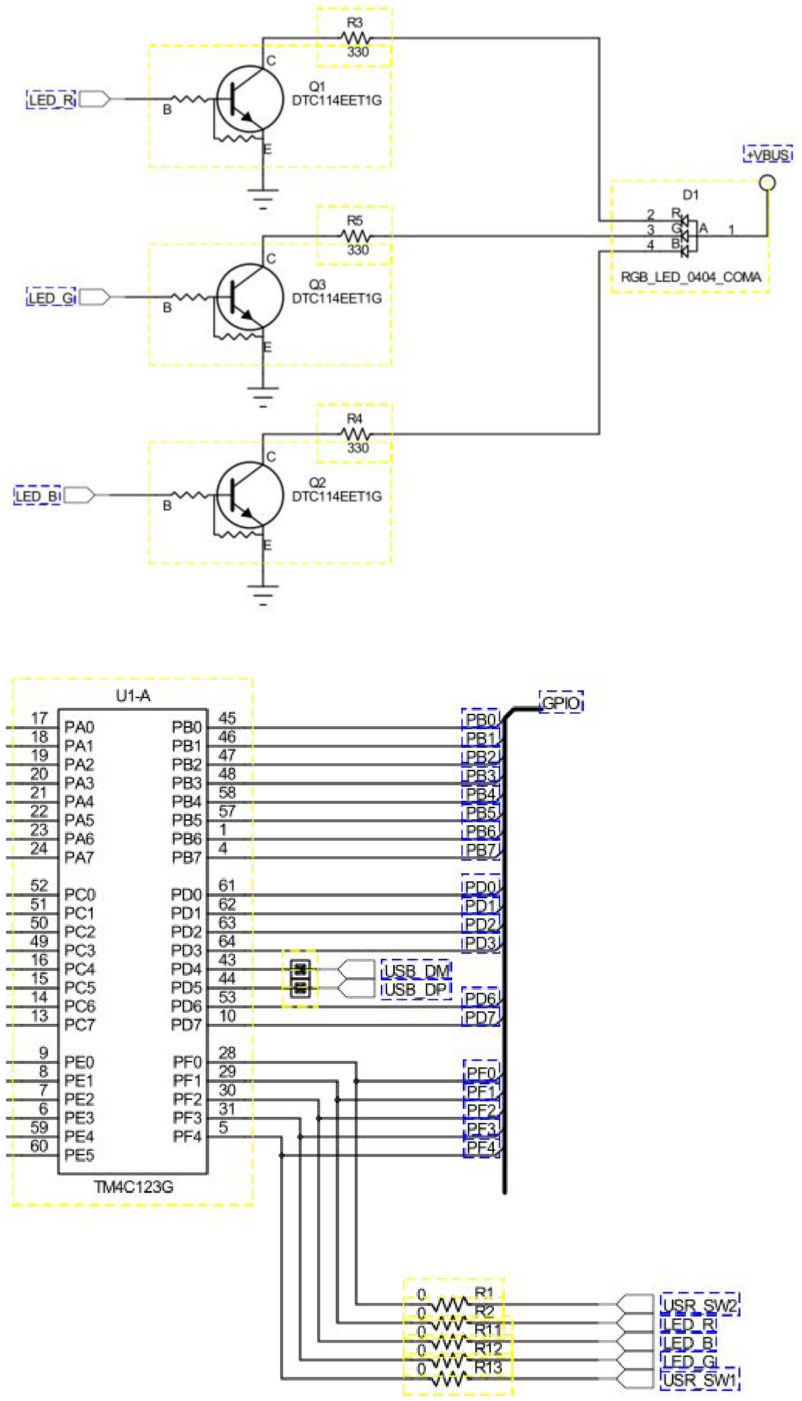
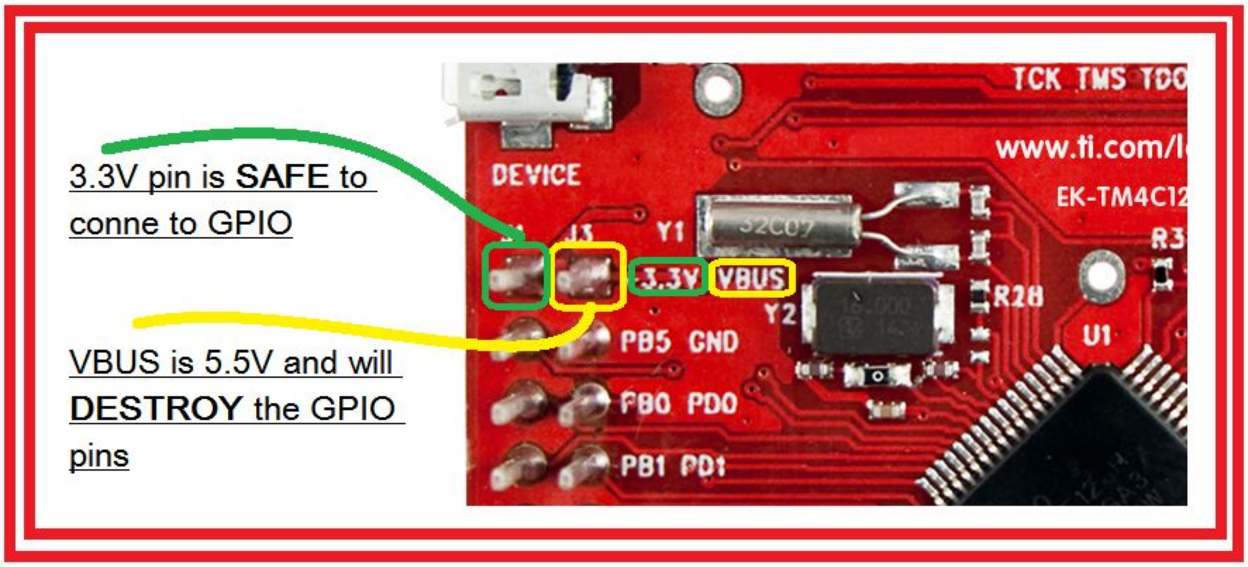


Figure 2. Tiva LaunchPad user LED connection schematics

Important: the pins on this board (like many other microcontroller boards) are **NOT** 5V tolerant. This means that they will be **destroyed** if you connect them to any 5V output (or, specifically, anything higher than ~3.6V). This means that connecting the VBUS pin to any other pin **will destroy that pin**.



IAR Embedded Workbench

IAR Embedded Workbench is a capable IDE for embedded development, and it is what we will be using in this course. It is installed on the computers in ECE 345, and instructions for installing it on you personal computer can be found under Canvas -> Files.

- IAR Workbench only supports Windows. If you're using Linux or Mac OS, you'll want to install a virtual machine. You could also use another IDE that supports the TM4C123, but this is not recommended.
- The free version of IAR Workbench that you will be downloading limits your code size to 32 KB. This should not be an issue for the assignments in this course.

Getting started

First, follow the tutorial found on the Canvas webpage for creating a new project in IAR Embedded Workbench.

Next, we're going to turn on the red LED on the TIVA development board. As shown previously, this LED (as well as the green and blue LEDs) are connected to Port F. To turn the LED on, we need to configure the values stored in the appropriate registers. These registers are located in the memory of the ARM Cortex M-4 on the development board. We can find information about all of the registers and their functionality from the TM4C123 datasheet. This datasheet will be very useful in this course, so it's a good idea to familiarize yourself with it. Note that it is *very* long.

To turn on the LED, we need to do the following:

1. Configure the **RCGCGPIO** register to turn on the GPIO Port F.

    - You may notice, by looking up the RCGCGPIO register in the data sheet, that this register can turn on or off any GPIO port. It does so by enabling or disabling a connection to the clock that would run them. By default, the clock to all peripherals, including GPIO ports, are turned off in order to improve the energy efficiency.
2. Configure the **GPIODEN** register to set Port F as digital input/output
    - None of the GPIO pins are set to be digital inputs/outputs by default**.**
3. Configure the **GPIODIR** register to set the direction of the Port F pins PF1, PF2, and PF3 to be outputs.
4. Configure the **GPIODATA** register to set the **output value** of the corresponding GPIO pin to 1. To turn on the red LED (as an example), PF1 must be set to 1 so the value of the port is "0x2" (in hexadecimal) or "0b0010" (in binary).
    - Note that since we've set the GPIO pins to be outputs, we **set** the value of the data register to control the outputs. If the pins were instead set to be inputs, we would **read** the incoming data from this register.

The following code does all of these steps and turns on the red LED.

```
#include <tm4c123gh6pm.h>
#include <stdint.h>

#define RED 0x2

int main()
{

  SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF; // enable Port F GPIO

  GPIO_PORTF_DIR_R = RED;   // set PF1 as output
  GPIO_PORTF_DEN_R = RED;   // enable digital pin PF1
  GPIO_PORTF_DATA_R = RED;  // set PF1 to 1 and the other port F pins to 0

  while(1)
  {

  }

  return 0;
}
```

Copy this code to the main.c file in the IAR Embedded Workbench project you created and download it to your board and debug it. To do this, click the green "start" icon at the top of the screen, or press Ctrl + D. Then, hit the white "start" icon at the top of the screen, or press F5.

The red LED should light up.

Note: If the red LED failed to light up, or the code failed to upload, please refer to the *TIVA Board Troubleshooting* file under Canvas -> Files -> Tutorials.

User Switches

Reading from the user switches is similar to writing to the LEDs. The difference is you'll have to set the direction of their pins (PF0 and PF4) to be inputs using the GPIODIR register and attach pull-up resistors.

Note:

- To attach pull-up resistors to pins PF0 and PF4, you'll need to use the **GPIOPUR** register.
- Before you can use the GPIOPUR register, the **GPIOCR** register must be configured correctly.
- Before the GPIOCR register can be configured, Port F needs to be unlocked using the **GPIOLOCK** register.

The datasheet has the information you need.

Assigned Task 1

Modify your existing project to achieve the following:

1. **Remove the inclusion of the tm4c123gh6pm.h header file.**
   Instead, make your own header file by creating a new file in IAR Embedded Workbench and saving it with the .h extension. To add your new header file to the project, right click in the "Files" area of IAR Workbench, go to "Add"
   In this file, define pointers to any registers you need to use.

   Take a look at the tm4c123gh6pm.h file to get an idea what this looks like. For example, the data register used in the code above is defined as follows:

   ```
   #define GPIO_PORTF_DATA_R       (*((volatile uint32_t *)0x400253FC))
   ```

   The correct addresses you need for all registers can be found in the datasheet, described by their base and offset.

2. **Modify the program to change the color of the LED in a periodic pattern.**
   First, you should figure out how to light up the blue and green LEDs. By turning on more than one color at the same time, you can easily create several other colors. In total, you'll be able to create 7 different colors by digitally writing to the 3 LEDs.

   Using loops, continuously change the color of the LED, going through all 7 colors. To create a delay between changing colors, you can use a loop. The following code introduces a delay of approximately 0.3 seconds:

   ```
   for (j = 0; j < 1000000; j++) {}
   ```

3. **Create a new program and turn on the LED using the user switches.**
   Your new program should turn on the red LED whenever switch 1 is pressed, and turn on the green LED whenever switch 2 is pressed. The location of these switches can be seen in Figure 1.

Assigned Task 2

For this task, you'll be designing a finite state machine to implement a traffic light system, and then implementing it using your TIVA launchpad and the provided peripheral LEDs and push buttons. Since these are external to the board, note that you will need a breadboard, and jumper wires.

External LEDs

In the first task, you used the onboard LEDs. Now, you'll need to utilize off-board peripheral LEDs. A green, yellow, and red LED can be found in your lab kit. You'll also find three $220\ \Omega$ resistors and a 74LS06 inverter chip. With these, you can create three copies of the circuit shown in Figure 3. This circuit will let you control an external LED by writing to the GPIO pin connected to the input of the inverter.

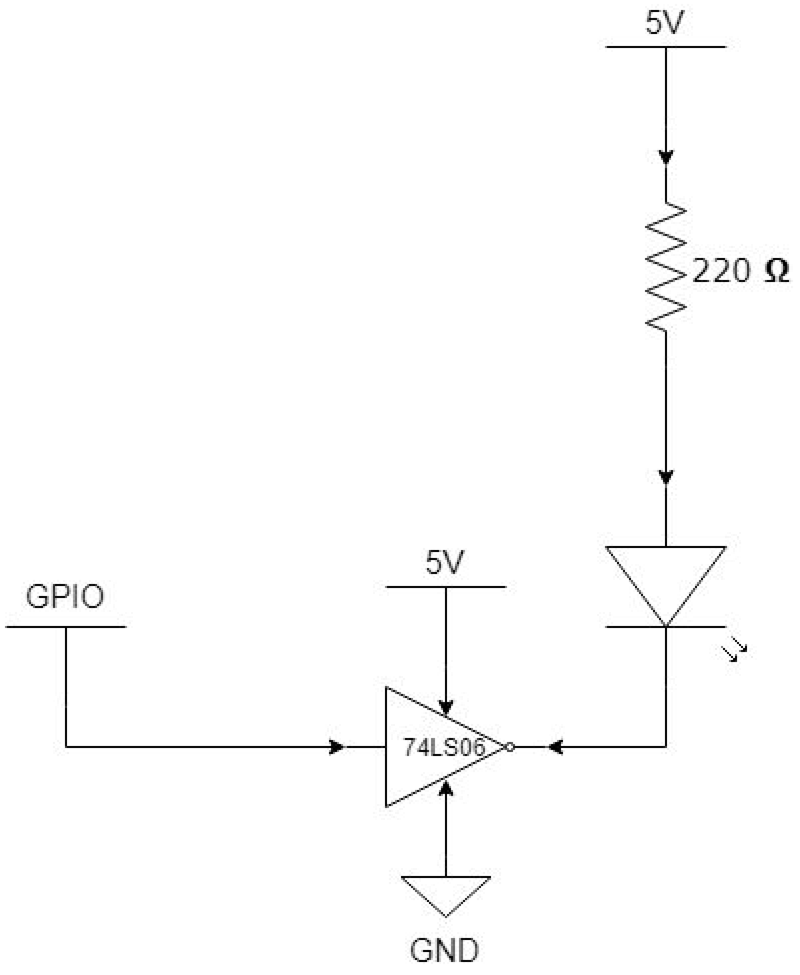


Figure 3. External LED circuit schematic

Recall that LEDs emit light when current passes through them, and as diodes, current will only flow from the anode (positive) to the cathode (negative). For LEDs, the polarity is indicated by the length of the pins – the positive anode pin is longer. Thus, in the circuit schematic above, the shorter leg of the diode should be connected to the output of the inverter chip, while the longer leg should go to the resistor.

External LED code

The following code snippet will enable you to turn on an LED when the signal marked “GPIO” in the above circuit schematic is connected to pin PA2:

```
void LED_init(void)
{
  volatile unsigned long delay;
  SYSCTL_RCGC2_R |= 0x01;              // activate clock for Port A
  delay = SYSCTL_RCGC2_R;              // allow time for clock to start
  GPIO_PORTA_PCTL_R &= ~0x00000F00;     // regular GPIO
  GPIO_PORTA_AMSEL_R &= ~0x04;         // disable analog function of PA2
  GPIO_PORTA_DIR_R |= 0x04;            // set PA2 to output
  GPIO_PORTA_AFSEL_R &= ~0x04;         // regular port function
  GPIO_PORTA_DEN_R |= 0x04;            // enable digital output on PA2
}
// turn on LED connected to PA2
void LED_on(void)
{
  GPIO_PORTA_DATA_R |= 0x04;
}
// turn off LED connected to PA2
void LED_off(void)
{
  GPIO_PORTA_DATA_R &= ~0x04;
}
```

You will need similar code to control all 3 of your LEDs.

External Buttons

Just like the LEDs, in this task, you will utilize off-board buttons instead of the buttons on the board. Buttons are simple: when the button is not being pressed, the two ends are disconnected. Once pressed, the two ends are connected in a short circuit.

To read from a button, connect one end to the 3.3V output of your TIVA board. Connect the other end to 1) the GPIO you’ll be reading from, and 2) a resistor connected to ground.

When the button is not being pressed, the GPIO will read LOW due to the connection with the resistor to ground. When the button is pressed, the GPIO will read HIGH from the 3.3V output.

External Button Code

The following code snippet will enable you to read from a button connected (as described above) to pin PA5.

```
void switch_init(void)
{
  volatile unsigned long delay;

  SYSCTL_RCGC2_R |= 0x00000001;         // activate clock for Port A

  delay = SYSCTL_RCGC2_R;               // allow time for clock to start

  GPIO_PORTA_AMSEL_R &= ~0x20;          // disable analog on PA5
  GPIO_PORTA_PCTL_R &= ~0x00F00000;     // PCTL GPIO on PA5
  GPIO_PORTA_DIR_R &= ~0x20;            // set PA5 to input
  GPIO_PORTA_AFSEL_R &= ~0x20;          // PA5 regular port function
  GPIO_PORTA_DEN_R |= 0x20;             // enable PA5 as digital port
}

unsigned long switch_input(void)
{
  return (GPIO_PORTA_DATA_R & 0x20); // 0x20 (pressed) or 0 (not pressed)
}
```

You will need code like this to read inputs from two separate buttons.

Traffic Light Controller

1. **Design a finite state machine representing the traffic light system described below.** Draw a block diagram for the system and write the C code that will implement it on your TIVA LaunchPad.

   System inputs:

   1. Start/Stop button
      The user will press this button to start and stop the system. When the system is started, it will always start in the *stop* state or the *go* state
   2. Pedestrian button
      The user will press this button to indicate that there is a pedestrian that would like to cross the street.
      - If the button is pressed in the *stop* state, nothing will happen.
      - If the button is pressed in the *go* state, the state should change to the *warn* state, and then change to the *stop* state.

   System outputs:

   1. Red LED
      This LED is on when (and only when) in the *stop* state
   2. Yellow LED
      This LED is on when (and only when) in the *warn* state
   3. Green LED
      This LED is on when (and only when) in the *go* state

When the system is started and the pedestrian button is not pressed, it should switch between the *go* state and the *stop* state.

3. **Implement your finite state machine using external peripherals**
   To implement your traffic light controller, you need to use two buttons and three LEDs. The following table lists some suggested pins to use to connect all of these components:

LEDs/Buttons	Possibility 1	Possibility 2	Possibility 3
Start/Stop Button	PA2	PB2	PE0
Pedestrian Button	PA3	PC4	PE1
Red LED (Stop State)	PA7	PB5	PE5
Yellow LED (Warn State)	PA6	PB4	PE4
Green LED (Go State)	PA5	PB3	PE3

Lab Demonstration / Submission

Lab Demonstration

Demonstrate your design for the assigned tasks to the TA.

Submission

Submit a short lab report that includes the following sections:

1. Procedure
   - Describe how you approached the assigned tasks and include any relevant diagrams, schematics, etc.
2. Results
   - Describe the results obtained for the assigned tasks. Give a brief overview of the finished product, as it relates to what was asked of you.
   - Include any relevant images, screenshots, etc.
3. Problems Encountered and Feedback
   - Describe any issues you had while completing the lab, and how/if you were able to overcome them.
   - Include any feedback you have about the lab, or any tips and tricks that you learned that may be useful to you in future labs.
   - Include how many hours you spent in completing this lab.
4. Appendix
   - In your Appendix, include any code you wrote in the lab. This should be **formatted** and **commented.**

A good resource for formatting code for Word documents is found here: http://www.planetb.ca/syntax-highlight-word

Additionally, submit your commented .c and .h files.

Grading

Your grade for this assignment will depend on the following:

- Lab demonstration: **50%**
    - Your lab demonstration will be graded based on your completion of the assigned tasks, your ability to answer questions, and your understanding of the course material.
- Lab report: **20%**
    - Your lab report should be short and concise.
- Code comments: **30%**
    - Comment your code! Your code should have function comments and in-line comments where appropriate.